Performance Report

StudentRegistry

**Structure Used**: HashMap  
**Time Complexity**:

* Lookup: O(1)
* Insert: O(1)
* Delete: O(1)

**Space Complexity**:

* O(n), where n is the number of students.

**Memory Estimate**:

* Each student record uses ~32–64 bytes (ID + name + object overhead).
* For 1,000 students: ~50–100 KB.

**Tradeoffs & Alternatives**:

* **LinkedList**: O(n) lookup — inefficient for frequent access.
* **TreeMap**: O(log n) lookup — slower than HashMap.
* **HashMap chosen** for constant-time access and minimal memory overhead.

CourseScheduler

**Structure Used**: ArrayList for enrolled students, Queue for waitlist  
**Time Complexity**:

* Enroll: O(1)
* Waitlist enqueue: O(1)
* Retrieve list: O(n)

**Space Complexity**:

* O(m), where m is the number of students per course.

**Memory Estimate**:

* Each course stores ~2 lists. For 100 courses with 50 students each: ~500 KB.

**Tradeoffs & Alternatives**:

* **Circular Array**: Fixed size, harder to resize dynamically.
* **Set**: No duplicates, no order — unsuitable for waitlists.
* **ArrayList + Queue chosen** for dynamic sizing and order preservation.

FeeTracker

**Structure Used**: Binary Search Tree (custom)  
**Time Complexity**:

* Insert: O(log n) average
* Report (in-order traversal): O(n)

**Space Complexity**:

* O(n), where n is the number of payments.

**Memory Estimate**:

* Each node ~48–64 bytes. For 2,000 payments: ~100–150 KB.

**Tradeoffs & Alternatives**:

* **AVL Tree**: Guaranteed balance, but higher overhead.
* **HashMap**: No chronological order.
* **BST chosen** for simplicity and natural date ordering.

LibrarySystem

**Structure Used**: HashMap + Deque for borrow history  
**Time Complexity**:

* Borrow: O(1)
* Return: O(1)
* Check availability: O(1)

**Space Complexity**:

* O(b), where b is the number of books.

**Memory Estimate**:

* Each book ~64–128 bytes. For 500 books: ~50–100 KB.

**Tradeoffs & Alternatives**:

* **HashMap-only**: No borrow history.
* **List**: Poor LIFO behavior.
* **Deque chosen** for efficient stack-based tracking.

AnalyticsModule

**Structure Used**: HashMap> + PriorityQueue  
**Time Complexity**:

* Record grade: O(1)
* Top-k ranking: O(n log k)

**Space Complexity**:

* O(s × c), where s is students and c is courses.

**Memory Estimate**:

* Each grade entry ~16–32 bytes. For 1,000 students × 5 courses: ~100–200 KB.

**Tradeoffs & Alternatives**:

* **Matrix**: Sparse, memory-wasteful.
* **Graph**: Overkill — no relationships to model.
* **HashMap + Heap chosen** for flexible storage and efficient ranking.

|  |  |
| --- | --- |
| component | Estimated memory space |
| Student registry | 100KB |
| Course scheduling | 500KB |
| Fee tracking | 150KB |
| Library system | 100KB |
| Analytics module | 200KB |
| total | 1-2MB for 1,000 students,100 courses |

Performance justification

* All modules use **dynamic, efficient data structures** suited to their tasks.
* **HashMaps** provide fast access and flexible storage.
* **BSTs and heaps** offer ordered retrieval and ranking.
* **Stacks and queues** model real-world behavior (borrowing, waitlisting).
* Alternatives like matrices, graphs, or fixed arrays were rejected due to poor fit, complexity, or memory waste.